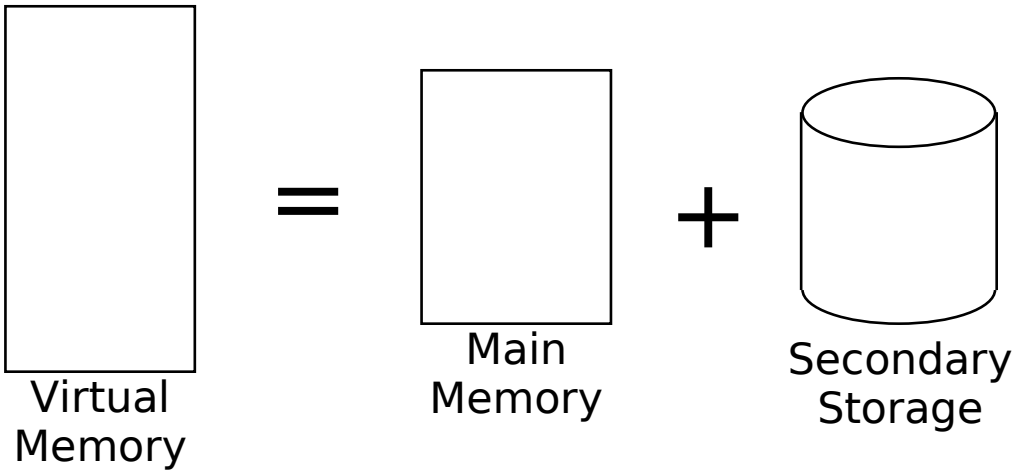
# Virtual Memory



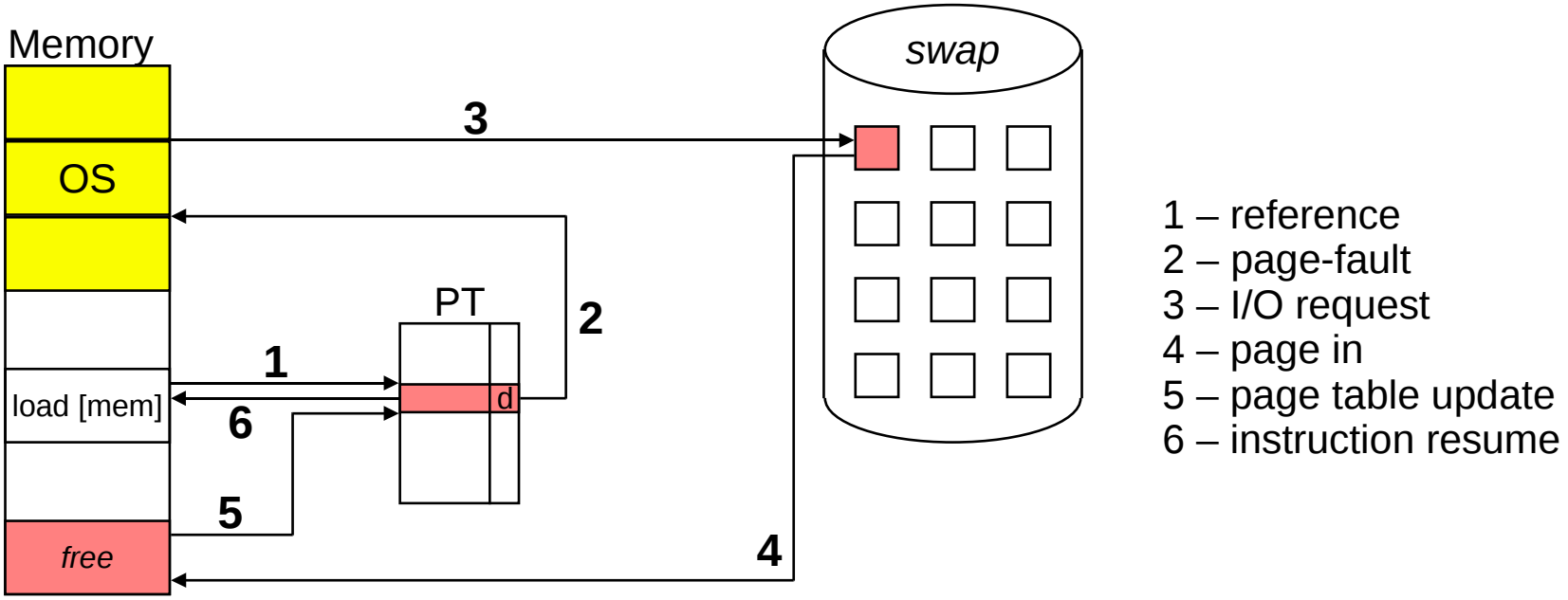

- Allows a process to be executed even if not completely loaded into memory
- Allows for processes to allocate more memory than the size of physical memory
- Can improve CPU utilization
- Can reduce swap overhead
- Can kill your system!

# Swapping

- Processes can be temporarily suspended and the memory allocated to them is first copied to a secondary storage (i.e. disk) and than released to other processes (*swap out*)
  - Sleeping processes
  - Low-priority processes
- Such processes can be latter resumed by restoring their address spaces from the copy in the secondary storage (*swap in*)

# Demand Paging

- Page-oriented swapping
  - Page table flag indicates whether the page is in memory or on disk
  - MMU triggers an exception (*page-fault*) whenever an absent page is accessed



# Page-Fault Handling

| | |
|---|---|
| Page-fautl trap | 1 µs |
| Save context | 10 µs |
| Dispatch PF handler | 1 µs |
| Locate page on disk | 50 µs |
| Read page from disk | 10 ms |
| Waiting queue | 0 s |
| Seek | 7 ms |
| Latency | 2 ms |
| Transfer | 1 ms |
| Scheduler | 15 µs |
| Disk I/O completion | 1 µs |
| Save context | 10 µs |
| Dispatch disk I/O handler | 1 µs |
| Update page table | 15 µs |
| Ready queue waiting | 0 s |
| Scheduler | 15 µs |
| TOTAL | 10,109 ms |

# Demand Paging Performance

- Formula

```
eat = (1 - p) x mat + p x pft
```

eat = effective access time
mat = memory access time
pft = page-fault handling time
p = page-fault probability

- Example

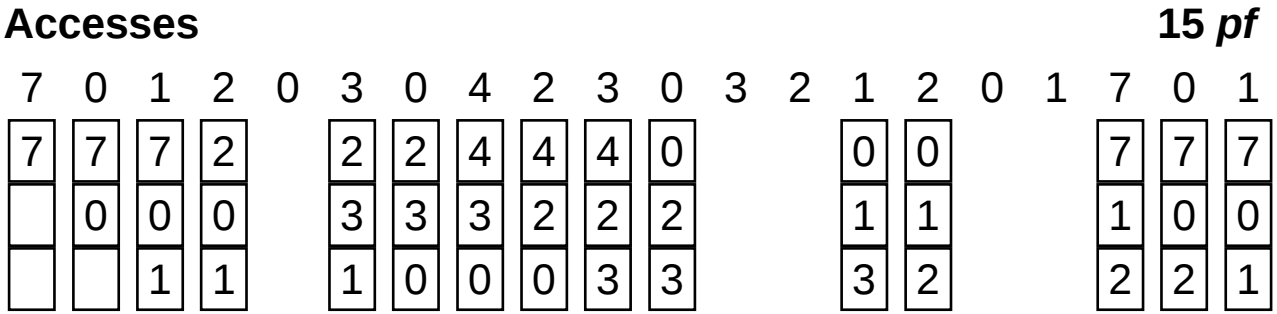
mat = 50 ns
pft = 10 ms = 10.000.000 ns
eat = (1 - p) x 50 + p x 10.000.000
eat = 50 + 9.999.950 x p
p = 0.001 -> eat = 10 ▪ s
eat = 50 ns -> p < 0,000.005 (1 / 200.000)

# Page Replacement

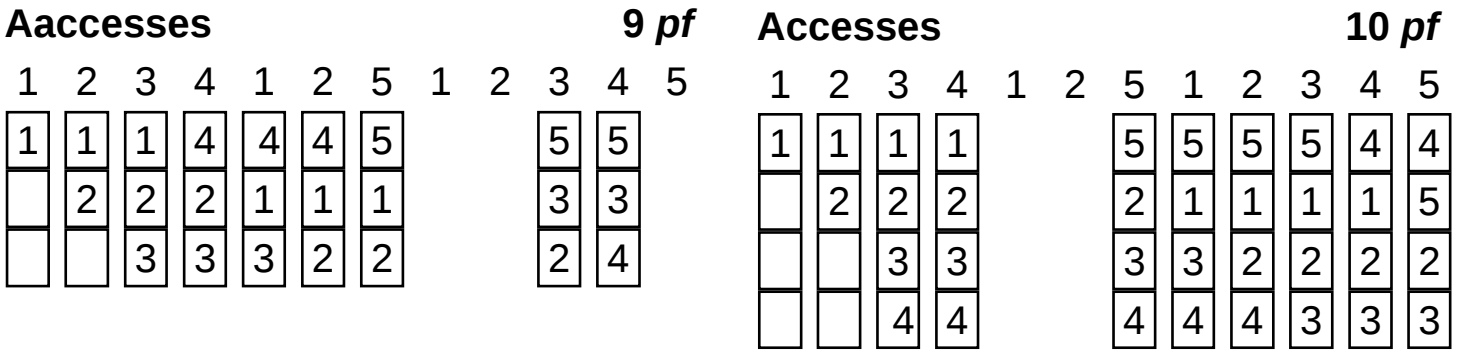
- Whenever necessary, OS selects pages to be moved out to disk and than make them available to processes
- Algorithm criteria
  - Minimize page-fault ratio
  - Minimize I/O
- Dirty bit
  - Each page status (modified or not) is kept in the corresponding entry in the page table
  - MMU automatically sets that bit whenever a page is modified (written)
  - Modified (dirty) pages must be written to disk before being reused

# First-In First-Out (FIFO)

- Replace the page that has been longer in memory (e.g. pages are time-stamped)
- Implemented using a FIFO queue
- Example

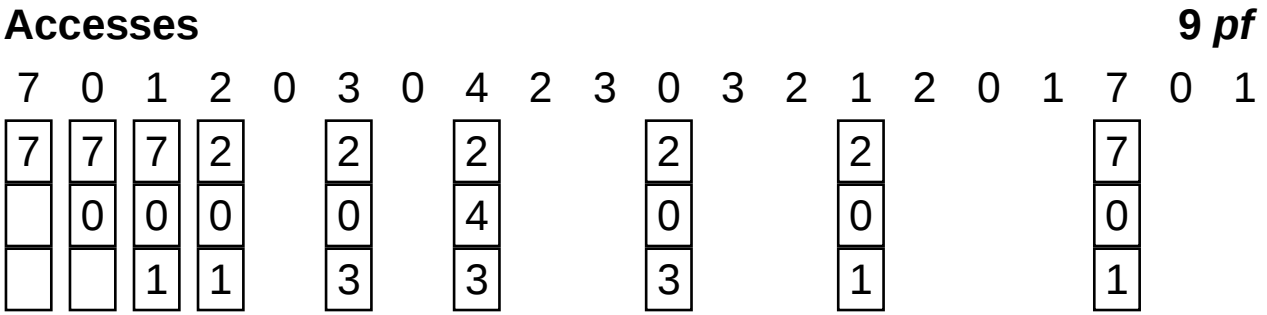



- Increasing memory may not decrease pf rate



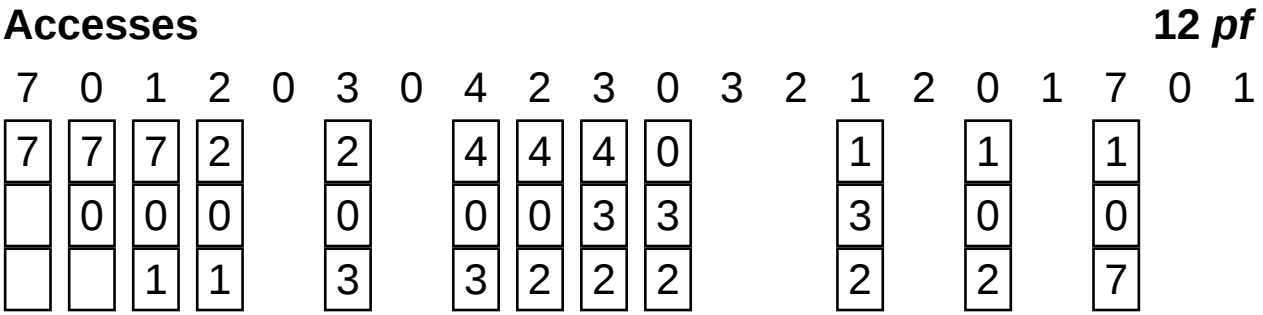
# Optimal

- Replace the page that will not be used for the longest period of time
- Not implementable, for it relies on knowing the future
- Example



# Least Recently Used (LRU)

- Uses the recent past as an approximation of the near future
- Replace the page that has not been used for the longest period of time
- Example




- Implementation
  - Time-stamp for each page
  - Linked-list of pages

# LRU Approximations

- Reference bit
  - Each page is assigned a reference bit that is set by the MMU whenever the page is accessed
  - OS clears those bits periodically
  - Order of use is unknown
  - Target page is any with cleared reference bit
- Reference word
  - Additional reference bits that are shifted by the MMU
  - Target pages are those with smallest reference values
- Second chance
  - Pages are tracked by a circular FIFO list
  - If the pointed page has a clear reference bit, it is taken to be replaced
  - Otherwise, the bit is cleared and the pointer is adjusted to the next page

# LRU Approximations

- Least Frequently Used (LFU)
  - Uses a reference counter for each page that is incremented by the MMU
  - Target page is the one with smallest counter value
  - Pages intensively accessed in the past, but no longer in use will take long to be replaced
- Reference and Modification bits
  - In addition to accesses, MMU marks pages that have been modified
  - Replacement order
    - Not-accessed, not-modified
    - Not-accessed, modified
    - Accessed, not-modified
    - Accessed, modified

# Allocation of Frames

- Minimum set of frames
  - Instructions and operands may be scattered across several pages
  - Architecture-dependent
  - Instructions must be restarted after a page-fault
- Frames per process
  - Proportional to process size (i.e. memory footprint)
  - Equal to all processes
- Process interference
  - A process might cause the replacement of a page initially allocated to other process
  - A process may only replace its own pages

# Thrashing

- A process is “thrashing” if it spends more time replacing pages the executing
- Causes
  - OS monitors CPU utilization and allows more processes in
  - If CPU utilization was low do to page-faults, increasing the number of processes in the system might cause thrashing
  - Thrashing only occurs if global page replacement (i.e. from other processes) is allowed
- Prevention
  - At any given time, a running process must have a set of pages available that fulfills its demands: its working set of pages (time x space locality)

# Final Considerations

- Process load
  - On-demand
  - At-once to main memory
  - At-once to swap disk
- Page size
  - Large -> less page-fault, less I/O, less page tables
  - Small -> less internal fragmentation
- I/O results
  - Pages that will recieve I/O results must be pinned-down
- Programming and code generation
  - Although virtual memory is functionally transparent to programs, memory access patterns might have big influence on it (e.g. matrices)